Concurrent Defects, Methodologies and Recommendations

Sasmita Padhy, Akash Kumar Sahu, Susanta Kumar Das

Abstract: ***Software testing is a major process in every software development cycle so as to produce superior quality products that can cater to the customer needs. In the beginning of the IT industry testing was a simple process since competition was not enough so as to produce good quality software. With the development of technology and fierce competition over recent years the needs to develop simultaneous methods of testing have been proposed. Reviewing the given application remains one of the major setbacks of concurrent code and the other being data flow in given request stack. Testing becomes really difficult when the function not returning the output to the caller function in requisite time but later on returns it via call-back functions, messages or other such processes. So this paper aims at viewing the tools and techniques available for better testing, removing bugs so as to make software good. Here the defects as well as directions need to remove it are also focussed.***

Keywords Used: Simultaneous, Concurrency, Testing, Tools, race conditions, dynamic analysis, static analysis

I. INTRODUCTION

Multi Threaded coding for concurrent applications have shown great progresses with advancement of microprocessor technologies. So new testing methods have been adopted as it requires existing software to test accordingly. With growth in research testing is finally able to meet its needs. Since concurrent applications lack new test methods for software, the hardware industries are speeding up the process in this direction by developing new hardware tools that can achieve concurrency power effectively.

There are a number of factors which are responsible for difficulty in concurrent testing such as:

- **Non determinism**- Absence of deterministic method makes it hard to detect bugs.
- **Concurrency Defects**-Structure of software, its blueprint, and application defects makes it concurrency failure.
- **Synchronization**: Integrating the components for concurrent execution for timing and sorting for execution.
- **Communication**: Execution of the application concurrently breach spatial isolation of components.

II. CONCURRENCY DEFECTS

One of the major drawbacks of system testing is the lack of knowledge of concurrency defects by software developers who unknowingly assume that the OS will take care of this. Some of such defects are:

a. Race conditions:

It happens mainly when in a multi threaded programming more than one thread access the same data and minimum one thread at that time is at write mode. [1] Race conditions are hard to detect in a code as it appears uncertainly in a different sections of the code at different times. But it can be avoided by sequencing operations between threads using synchronization.

b. Deadlock:

It was introduced to avoid race conditions. It is a condition where a process is blocked since each thread is holding a resource and waiting for another resource to be acquired by the some other processes. Deadlock[2] is most probable in processes where there is no circular wait conditions.

c. Starvation:

It is caused when in a multithreaded application the runnable codes are blocked or delayed. Starvation[3] is caused because of OS scheduling rules. To control starvation, OS scheduler timely interferes to boost priority of starving threads.

d. Livelocks:

Livelock[5] occurs when scheduled threads are not able to move forward because of their continuous reactions to state changes. One of the best examples of livelock is the giant utilization of CPU with minimal work done. Livelock becomes very hard to detect and repair.

e. Suspension:

It occurs when concurrent components are made standby for long time to acquire shared resources.

f. Atomicity Violation:

Atomicity causes crash in the system as processes interrupt with each other for execution.

g. Priority Inversion:

This causes failure of the process since lower priority operations are executed earlier as compared to higher priority.

III. CONCURRENT TESTING METHODOLOGIES

Concurrent software are tested to measure their performance, quality as well as their correctness which is resolved using methods like dynamic analysis,

Revised Manuscript Received on February 25, 2020.

Sasmita Padhy, department of Computer Science, VITAM, Berhampur, Odisha. Email:-pinky.sasmita@gmail.com
Akash Kumar Sahu, department of Computer Science, NIST, Berhampur, Odisha. Email:-akashkumarsahu987@gmail.com
Susanta Kumar Das, department of Computer Science, Berhampur University , Berhampur, Odisha. Email:-dr.s.k.das1965@gmail.com

static analysis and model checking.

A. Race free-typed method

It is a technique which implements a type based system to remove the race around conditions from program[2] as it does not suffer any performance sanctions. The major setback in this method is the expressiveness of applied type system which limits the coder's willingness to implement techniques he is used to. The benefits are it protects the shared resources using locks.

B. Static analysis method

It is generally implemented by seeing at the metadata in a executed programme or annotated source code [4], [5]. It generally analyzes the program without even compiling. Some of the popular tools used are the Prefast and Prefix, FxCop etc. It consists of formal inspection to avoid unnecessary behaviour.

Flow sensitive used in this method to encounter data race and deadlocks are figured in [6].

Advantages of static tool are:

- Complete coverage is gained.
- Easily able to detect bugs and fix it through precision.
- Coders gain confidence for formal analysis to build software.

Disadvantages includes:

- A significant amount of annotations are required to deal with concurrency bugs.
- A great amount of effort needs to be applied so as to reduce false positives generated during testing.

C. Dynamic analysis method:

Detection of bugs are done using program footprints. It only follows the visible path that has exact image of shared resources and executes on runtime.

It is done in online as well as offline mode. Analyzing as well as executing is done simultaneously in online mode whereas in offline mode first it traces them and analyzes later for bugs. It is more convenient technique as it generates minimum false positives and needs very less time for developers during coding. It is a low cost reliability method since the errors are detected in the most frequent executed path.

The disadvantages of dynamic analysis include:

- Errors are detected only along executed path and it needs reliability on test values for maximum coverage.
- Since finding the race conditions is not certain in some tools, detecting bugs sometimes is not possible.
- As the tools rely on some hardware that manipulates the runtime, its performance can be poor.

There are basically two main algorithms focused in [8].

- **Lockset Algorithm:**

Tools using this algorithm have a potential race which happened when two threads have access to shared memory resource without interleaving of threads which provides conditions for race potential. This technique does not generate false negative. But it can generate superfluous false positive . So it has poor precision.

- **Happen-before Algorithm:**

Here potential race occurs when two threads have a shared memory location. Here accesses are randomly sorted as in [9]. It produces less false positives as compared to lockset algorithm but it is difficult to implement.

D. Model checking method:

Reference[5] suggests this technique for correcting limited state of concurrent system. This method allows for formal deductiveness by trying to replicate both race and deadlock conditions. It gives maximum coverage with minimum connected hardware and provides higher confidence in structural design. But it has certain disadvantages which includes difficulty in model extraction from code due to excess verification of state space explosion where probable state count is huge. So using the reduction technique the state space explosion is controlled. Here there are certain constraints like failure in detection of bugs. Model checking ensures that application is error free whereas there can still be error in the implementation. It needs efficient architecture and planning which makes which makes it suitable for small portions in a product.

E. Hybrid method:

[12]It is the combinations of more than one technique so as to avail the benefits of each other. This combination can be unidirectional as well bidirectional.

- **Unidirectional:**

Here in this technique information is transferred from one method to another. Here dynamic analysis is guided by static analysis to generate minimum false positive rates.

Bidirectional:

Here information is interchanged amongst each other as one technique provides first information to next one that uses it during mechanism and in return provides the first one with feedback and results. Here static analysis guides dynamic analysis regarding the efficiency of testing.

Also to control large software model checking and static analysis are combined forming an iterative method[11]. In his technique static method analyzes errorless statements to be used by model checker for partial order analysis.

F. Structural White Box unit and Integration testing Methods.

Here the main aim is to generate plenty of test cases to maximize coverage by using structural information to recognize synchronization followed by communicating points.

G. Random testing

In this method test suites are automatically generated based on system behaviour.

H. Defect-Driven Testing

Here test suites are generated manually based on structure and design of the system.

IV. CONCURRENCY TESTING TOOLS

There are a large number of concurrency testing tools for detecting deadlocks, livelocks etc.

CHESS[13] tool combining model checking and dynamic analysis used for detecting concurrency errors by analyzing schedules.

It can detect deadlocks, livelocks, data corruption issues. It also provides coverage. It is also used as a dynamic tool required running unit test on a specifically designed scheduler. The technique used in CHESS is partial order reduction and to control state space explosion iteration context bounding is used. Upon repetition it chooses new scheduling.

CHESS technique boundaries itself to amount of thread switches in compilation process. This method is based on the formula of switching sorted threads to find out the concurrent bugs. CHESS also detects deadlocks and race around but it solely relies on coders for verification and it also reports livelocks.

INTEL THREAD CHECK[10] tool based on dynamic analysis used to find deadlocks, stalls, false API etc. It uses the Happen-Before technique on partial order. Here the thread checker makes the memory reference by implementing the source code or compiled binary code.

But the tool has certain disadvantages as it cannot operate on interlocked operations on synchronization like custom spin locks. It is best for applications using standard synchronizations for concurrency testing.

RacerX[6] tool is used for analyzing race around and deadlocks. In this technique the only condition is that the user has to present a table requiring APIs so as to access and release locks. The table needs to be very small of less than 30 entries. It operates phase wise. First of all it loops over the initial code file building CFG (control Flow Graph) containing information of functions, shared memory, pointer usage etc. As the CFG is completed, the next stage of analysis of race checker as well as deadlock checking comes in. Proper caching methods to reduce traversing is used after which the Lockset algorithm applies for potential race. So in this way it repeats locking cycle when locks are used.

The last phase contains the post-processed errors so as to prioritize the negative effects of error. The results of this tool are amazing as it reduces false positives and removes bugs.

CHORD [7] is a static analysis tool which is context sensitive as well as flow insensitive which allows it to be more flexible as compared to other tools. Here the algorithm is a mess of concepts and it uses Java specific primitive synchronization.

ZING This tool has its own custom language and is a model checker used for design verification. It is fully able to model concurrent state machines. It uses innovative reduction technique for concurrent state space explosion and it provides a clear way to verify designs and building high quality applications. Here the model needs to be created by translator or manually. But without translator ZING cannot be used for large projects.

V. ANALYSIS RESULT

Concurrency tools	Methodologies	Techniques	Application
CHESS	Combining model checking and dynamic analysis used for detecting concurrency errors by analyzing schedules.	Partial order reduction and to control state space explosion iteration context bounding is used. Upon repetition it chooses new scheduling.	CHESS is good for developers who can take little stress to get better interleaved testing..
INTEL THREAD CHECK	Dynamic analysis used to find deadlocks, stalls, false API	Happen-Before technique on partial order	It is the best testing tool can be used for applications which are developed for a particular device.
RacerX	Static Analysis tool and Race free typed method used to find deadlocks	Lockset algorithm applies for potential race	From a test engineering perspective, the output of this tool is excellent and realistic.
CHORD	Static analysis tool which is context sensitive as well as flow insensitive which allows it to be more flexible as compared to other tools	Here the algorithm is a mess of concepts and it uses Java specific primitive synchronization.	Perhaps this can be the most-promoted testing tools available for device-specific applications
ZING	It uses own custom language and a model checker used for design verification	It uses innovative reduction technique for concurrent state space explosion and it provides a clear way to verify designs and building high quality applications	Zing tool can only check the correctness but to test for concurrent bugs it need a translator for large applications

VI. RELATED WORKS

In [14], [15] non deterministic testing been developed for execution of code with same value to find errors by repeated execution. But here lack of control may lead to SYN-sequence.

Other works [16], [17] include deterministic testing to exercise selected SYN-sequences from the graph of code. Here the problem lies in state explosions it selects totally ordered SYN-sequences of various definite partial orders.
Another technique contains both deterministic as well as non deterministic known as Reachability [18] which is designed for shared multi threaded programs. Later in [20] improvements have been made for asynchronous message passing. Recently semaphore based program [21] and monitor based programs[22] are developed for reachability testing[23].
One more is proposed in [24] so as to avoid race variant from being generated so as to have duplicate SR sequence. New tools like PathFinder [25], VeriSoft [26], ExitBlock [27], uses partial order reduction. But reachability testing uses half order directly. Also SYN-sequence is highly flexible as it is based on language level definition of concurrency rather than structure. Reachability testing requires no modifications but the above mentioned tools require access to thread scheduler.

VII. RECOMMENDATIONS

Commonly used recommendation is:

- **Study**: Deeply studying new concurrency defects and testing techniques.
- **Plan**: Planning for maximum coverage metrics is needed.
- **Check**: Need of checking interleaving of different operations of components
- **Review**: Structure and design must be reviewed for errors
- **Cover**: Various test cases need to be developed for interleaving.
- **Be attentive**: Need to pay attention for critical situations.
- **Peer Review**: Need of it in design and structure of application.

VIII. CONCLUSION

In this paper we concentrated on various strategies and methods for evaluating simultaneous and multi-threaded programmes. Static tools are simple and scalable and can run in broad real code but generate false alarms. Dynamic analysis methodology gives accurate outcomes but still it suffers from low faith since it only analyzes paths that were executed. Because of the pros and cons of different testing methods, modern research has been attempted to incorporate more than one method to maximize mutual advantage.

REFERENCES

1. T. A. Henzinger, R. Jhala, and R. Majumdar, “Race checking by context inference,” SIGPLAN Not., vol. 39, no. 6, pp. 1–13, Jun. 2004.
2. M. Naik, C.-S. Park, K. Sen, and D. Gay, “Effective static deadlock detection,” 2009, pp. 386–396.
3. V. P. Rahul and G. Boby, “Tools And Techniques to Identify Concurrency Issues.” [Online]. Available:http://msdn.microsoft.com/enus/ magazine/ cc546569.aspx. [Accessed: 25-Feb-2012].
4. A. Raza, “A Review of Race Detection Mechanisms,” in Computer Science – Theory and Applications, vol. 3967, D. Grigoriev, J. Harrison, and E. A. Hirsch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 534–543.
5. N. E. Beckman, “A Survey of Methods for Preventing Race Conditions,” 2006.
6. D. Engler and K. Ashcraft, “RacerX: effective, static detection of race conditions and deadlocks,” 2003, p.237.
7. P. Pratikakis, J. S. Foster, and M. Hicks, “LOCKSMITH: Practical static race detection for C,” ACM Trans. Program. Lang. Syst., vol. 33, no. 1, pp. 3:1–3:55, Jan. 2011.
8. R. O’Callahan and J.-D. Choi, “Hybrid dynamic data race detection,” 2003, p. 167.
9. L. Lamport, “Ti clocks, and the ordering of events in a distributed system,” Commun. ACM, vol. 21, no. 7, pp. 558–565, Jul. 1978.
10. S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, “Eraser: a dynamic data race detector for multithreaded programs,” ACM Trans. Comput. Syst., vol. 15, no. 4, pp. 391–411, Nov. 1997.
11. G. Brat and W. Visser, “Combining Static Analysis and Model Checking for Software Analysis,” PROC. ASE 2001, pp. 262–271, 2001.
12. J. Chen and S. MacDonald, “Towards a better collaboration of static and dynamic analyses for testing concurrent programs,” in Proceedings of the 6th workshop on Parallel and distributed systems: testing, analysis, and debugging, New York, NY, USA, 2008, pp. 8:1–8:9.
13. M. Musuvathi and S. Qadeer, CHESS:Systematic Stress Testing of Concurrent Software. 2006.
14. O. Edelstein, E. Farchi, Y. Nir, G. Ratsaby, and S. Ur. Multithread Java program test generation. IBM Systems Journal, Vol. 41(1), pp. 111-125, 2002.
15. S. D. Stoller. Testing concurrent Java programs using randomized scheduling. In Proceedings of the Second Workshop on Runtime Verification (RV), Vol. 70(4) of Electronic Notes in Theoretical Computer Science. Elsevier, 2002.
16. K. C. Tai. Testing of concurrent software. Proc. of the 13th Annual International Computer Software and Applications Conference, pp. 62-64, 1989
17. R. N. Taylor, D. L. Levine, and Cheryl D. Kelly, “Structural testing of concurrent programs”, IEEE Transaction on Software Engineering, 18(3):206-214, 1992.
18. G. H. Hwang, K. C. Tai, and T. L. Huang. Reachability testing: An approach to testing concurrent software. International Journal of Software Engineering and Knowledge Eng. 5(4):493-510, 1995.
19. K. C. Tai. Reachability testing of asynchronous messagepassing programs. Proc. of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems, pp. 50-61, 1997.
20. Y. Lei and K. C. Tai, Efficient reachability testing of asynchronous message-passing programs, Proc. 8th IEEE Int’l Conf. on Engineering for Complex Computer Systems, pp. 35-44, 2002.
21. Y. Lei and R. Carver, Reachability testing of semaphore based programs, Proc. of COMPSAC, 2004.
22. Y. Lei and R. Carver, Reachability testing of monitor-based programs, Proc. of Software Engineering and Applications, 2004.
23. R. Carver and Y. Lei, A general model for reachability testing of concurrent programs, to appear in Proc. of Intl. Conf. on Formal Engineering Methods, 2004.
24. Y. Lei and R. Carver, “A New Algorithm for Reachability Testing of Concurrent Programs,” Proceedings of the 16th IEEE International Symposium on software engineering ,2005
25. W. Visser, K. Havelund, G. Brat, and S. Park. Java PathFinder – Second Generation of a Java Model Checker. In Proc. of Post-CAV Workshop on Advances in Verification, 2000.
26. P. Godefroid. Model Checking for Programming Languages using VeriSoft. Proceedings of the 24th ACM Symposium on Principles of Programming Languages, pages 174-186, Paris, January 1997
27. D. L. Bruening. Systematic testing of multithreaded Java programs. Master’s thesis, MIT, 1999.

AUTHORS PROFILE

Dr. Sasmita Padhy belongs to Berhampur and borned on 10th June 1979. She received the B.E. degree in Computer Science from Utkal University, Odisha in 2001 and received M.Tech. in Computer Science from Biju Patnaik University of Technology (BPUT), Odisha in 2007. She also completed Ph.D. from Berhampur University, Berhampur, Odisha in 2012. Her area of interest is MIMO wireless communication and networks, wireless ad-hoc networks.

Akash Kumar Sahu, student of B.tech (2016-2020) in Computer science in NIST, berhampur. Her area of interest is MIMO wireless communication and networks, wireless ad-hoc networks, software engineering, software testing , system security.

Dr. Susanta Kumar Das joined the Dept. of Computer Science in 1993. He has teaching experience of 23 years in the department. He has attended no. of national & international conferences. To his credit, he has served as H.O.D for 2 years in the department. At present he is the coordinator of M.Tech (S.F) course & as coordinator of spoken tutorial project conducted by IIT Bombay & funded by MHRD, Govt of India. Fourteen no. of scholars are awarded Ph.D under his guidance. One D.Sc degree is awarded in Computer Science under his guidance. He has been felicitated award of honour by Dept. of Mathematics, Maharshi Dayanand University Rohtak, Haryana in the international conference on History & Development of Mathematical Science & Symposium on Nonlinear Analysis. His research are in Software Engineering & Network Security.